

Software Development Principles

Lecture 3 Definitions

Lecturer:

Karen Nolan

karen.nolan@it-tallaght.ie

Topics

- Introduction to Definitions
 - No Arguments
 - No Returns
- Definitions with Arguments
 - Single Arguments
 - Multiple Arguments

Python Definitions

- Writing code sometimes requires the reuse of code.
- This has two issues associated with it:
 - Time consuming
 - Erroneous due to copy and pastings code
- Also our current method of development does not allow for individual code to be developed and then brought together in a standardized practice
 - Could try and insert into code
 - Could take longer than writing code individually
 - Could lead to very erroneous code, with a large amount of bugs
 - Example: Variable naming schemes by two users may be very different

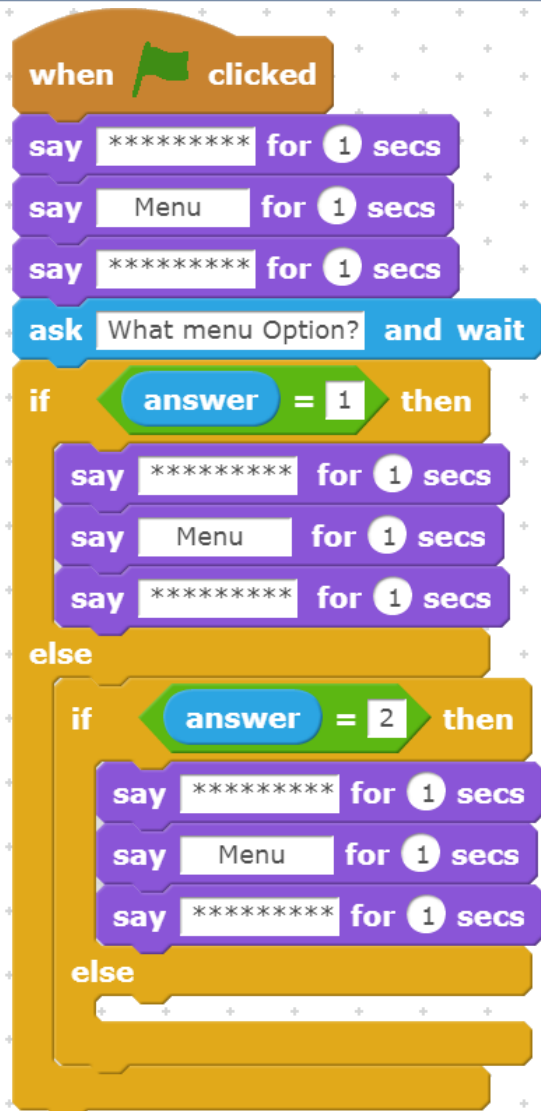
Example 1:

```
menuOption = 0
while menuOption != 3:
    print("\t*****")
    print("\t*           Menu           *")
    print("\t*****")

    menuOption = int(input("\tPlease enter menu
option:"))
    if menuOption == 1:
        print("\t*****")
        print("\t*           Menu           *")
        print("\t*****")
```

- There may be a case that Menu's may need to be printed multiple times.
- This introductory example, is a good use case for Definitions

Example 1:

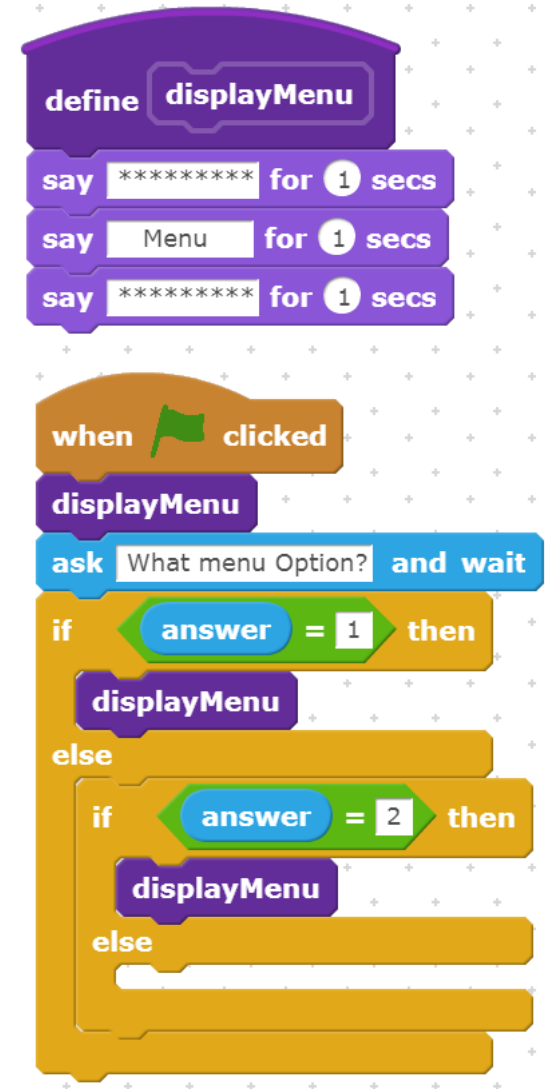


- Before we look at Python, the same problem can exist in all other languages, even Scratch!
- The below block is repeated multiple times.



Example 1:

- We can “Define” a block and use it many times in the code.
- The definition on its own, will not run.
- The definition must be called.
- Python must know that it exists.
- As Python runs as a script line by line, the definition ***MUST*** be written before the code that calls it.



Example 1:

- A Definition in Python is written as follows:

Definition



```
def display_menu():
    print("\t*****")
    print("\t*           Menu           *")
    print("\t*****")
```

Calling



Definition

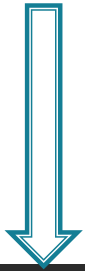


```
menuOption = 0
while menuOption != 3:
    display_menu()
    menuOption = int(input("\tPlease enter menu
option:"))
    if menuOption == 1:
        display_menu()
```

Definitions: Structure

- A Definition in Python is written as follows:

Structure Type



```
def display_menu():  
    print("\t*****")  
    print("\t*           Menu           *")  
    print("\t*****")
```


Definitions: Structure

- A Definition in Python is written as follows:

Definition Name (Convention is all lower case)



```
def display_menu():  
    print("\t*****")  
    print("\t*      Menu      *")  
    print("\t*****")
```

Definitions: Structure

- A Definition in Python is written as follows:

Definition Arguments (in this case, there are none)



```
def display_menu():  
    print("\t*****")  
    print("\t*           Menu           *")  
    print("\t*****")
```

If no arguments, you still require the ()

Definitions: Structure

- A Definition in Python is written as follows:

Definition Begin Similar to **for** and **while** & **if**



```
def display_menu():  
    print("\t*****")  
    print("\t*           Menu           *")  
    print("\t*****")
```

Definitions: Structure

- A Definition in Python is written as follows:

Definition Code

```
def display_menu():  
    print("\t*****")  
    print("\t*           Menu           *")  
    print("\t*****")
```

Uses indentation (Similar to **for** and **while** & **if**)

Definitions: Class Example 1

- Write a Python Definition that displays the value of π (22.0 / 7.0):
- Use a Loop to iterate 5 times to print out the value, using the definition:

Definitions: Class Example 2

- Write a Python Definition that displays the current time (including seconds)
- Use a Loop to iterate 5 times to print out the value, using the definition:

```
import time  
timeNow = time.strftime("%H:%M:%S")
```

Definitions: Arguments

- Useful to be able to “send additional data” to the Definition.
- When a definition **is** called, the **arguments are** the data you pass into the definition’s parameters.
- **Parameter** is the variable in the declaration of the Definition.
- **Argument** is the actual value of this variable that gets passed to Definition.

Note: Python does allow access to variables when using Definitions (unlike Java or C# where there is variable scope), we will use a mix of both for good practice.

Definitions: Arguments

- We can “pass” in a value to a Definition, inside the () brackets.
- We already do this on a regular basis

```
print("Hello")
```

- Lets print a value using an argument (our own version of print!)

```
def my_print(stringIn):  
    print(stringIn)
```

```
my_print("Hello")
```


Definitions: Arguments

- Lets step through the code:

```
1) def my_print(stringIn):  
2)     print(stringIn)  
  
3) my_print("Hello")
```

1) created a Definition with one argument => stringIn

Definitions: Arguments

- Lets step through the code:

```
1) def my_print(stringIn):  
2)     print(stringIn)  
  
3) my_print("Hello")
```

- 2) The code uses the Argument **stringIn** as a local variable
(Not visible outside Definition : Scope)

Definitions: Arguments

- Lets step through the code:

```
1) def my_print(stringIn):  
2)     print(stringIn)  
  
3) my_print("Hello")
```

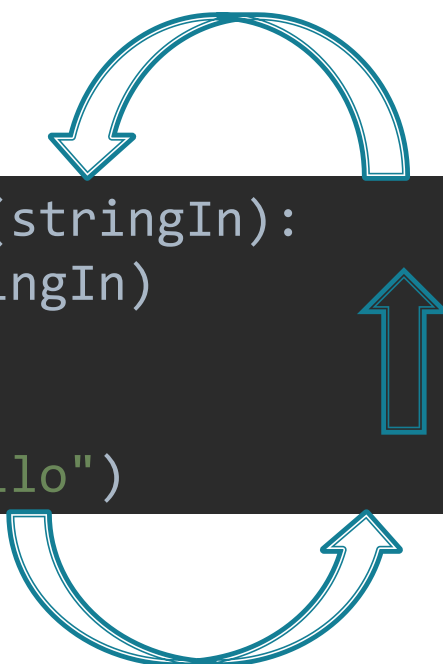
- 3) The code calls the Definition and sends the string “Hello” to the Definition as an argument. This is then locally stored as `stringIn`. Then on line two of the code, the print uses the local variable `stringIn` (with a value “Hello”), and prints it to the screen.

Definitions: Arguments

- Lets step through the code:

stringIn =

"Hello"



```
1) def my_print(stringIn):  
2)     print(stringIn)  
  
3) my_print("Hello")
```

Definitions: Class Example 3

- Write a Python Definition that has one Argument, a number, it squares the number and prints it to the screen.
- The user must enter the number to be passed into the Definition.

Definitions: Multiple Arguments

- The following Definition has two arguments:

```
def my_print(stringIn, numberIn):  
    print(stringIn * numberIn)  
  
my_print("Hello", 3)
```

- 1) created a Definition with two Argument's => stringIn, number

Definitions: Multiple Arguments

- The following Definition has two arguments:

```
def my_print(stringIn, numberIn):  
    print(stringIn * numberIn)  
  
my_print("Hello", 3)
```

- Definitions can have multiple Arguments
- Separated by a comma (,)
- They can be the same type or different types

Definitions: Class Example 4

- Write a Python Definition that has two Argument, both numbers.
- The user must enter both numbers to be passed in as Arguments.
- The Definition prints the largest of the two numbers.

Definitions: Separate .PY file

- A useful tool for reusability is separating your Definitions into a separate Python file.

myDefinitions.py



All of my Definitions



myMain.py



Main Python Script

Definitions: Separate .PY file

- A useful tool for reusability is separating your Definitions into a separate Python file.

myDefinitions.py



All of my Definitions

```
myDefinitions.py
1 def largest_number(number1_in, number2_in):
2     if number1_in > number2_in:
3         print("The largest number is :", number1_in)
4     elif number2_in > number1_in:
5         print("The largest number is :", number2_in)
6     else:
7         print("The two numbers are equal")
```

Definitions: Separate .PY file

- A useful tool for reusability is separating your Definitions into a separate Python file.

myDefinitions.py



myMain.py



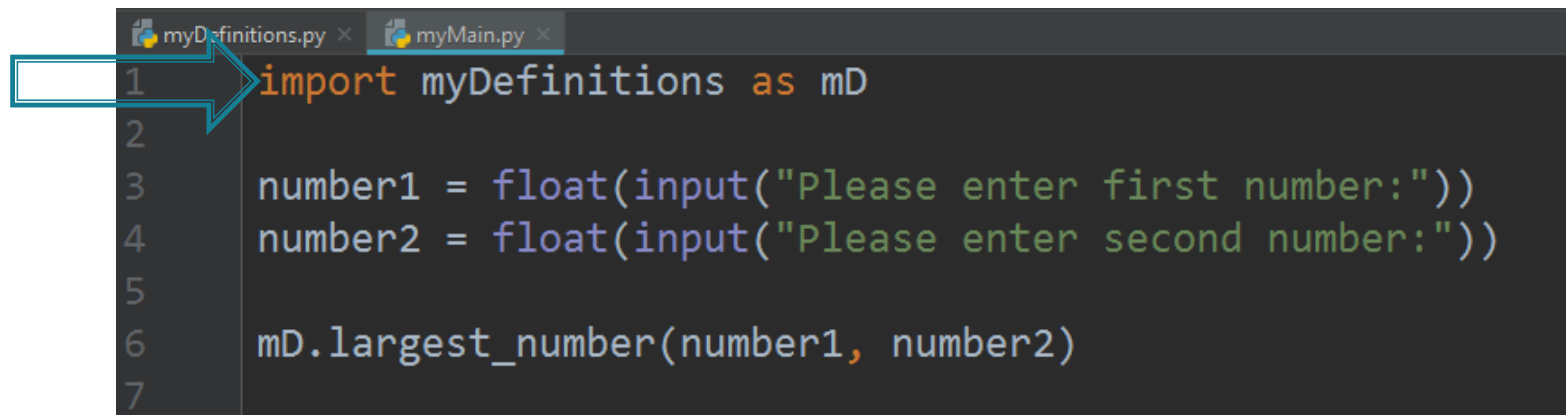
All of my Definitions

Main Python Script

```
myDefinitions.py x myMain.py x
1  import myDefinitions as mD
2
3  number1 = float(input("Please enter first number:"))
4  number2 = float(input("Please enter second number:"))
5
6  mD.largest_number(number1, number2)
7
```

Definitions: Separate .PY file

- A useful tool for reusability is separating your Definitions into a separate Python file.

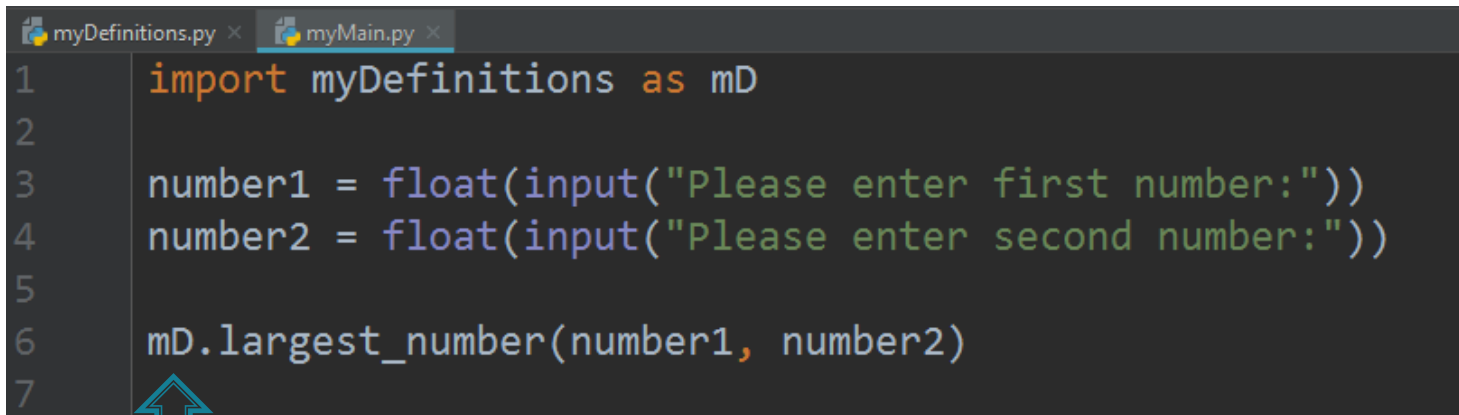


```
1 import myDefinitions as mD
2
3 number1 = float(input("Please enter first number:"))
4 number2 = float(input("Please enter second number:"))
5
6 mD.largest_number(number1, number2)
7
```

- Import, just like time, must use **as**

Definitions: Separate .PY file

- A useful tool for reusability is separating your Definitions into a separate Python file.



```
1 import myDefinitions as mD
2
3 number1 = float(input("Please enter first number:"))
4 number2 = float(input("Please enter second number:"))
5
6 mD.largest_number(number1, number2)
7
```

- Also need to use mD